Vinh Nguyen
300473488

Proposed Methodologies (Draft)

Overview:

We are trying to explore Evolutionary Reinforcement Learning (EvoRL) for the Flexible Job-shop Problem so I will only be focusing on applying the evolutionary techniques to the Deep Reinforcement Learning (DRL) part of the paper (Song et al, 2022). However, if the proposed methodologies prove to be insufficient or too difficult, other domains can be explored as well.

Baseline:

Techniques used in the paper:

- ***Actor-critic structure***
  (employs both value functions (critic) and policy search (actor) techniques)
- ***Model-free approach***
  (adjusts the policy for optimal rewards based on the outcomes of an action)
- ***On-policy learning***
  (learns the reward function from taking actions using the current policy)
- ***Loss function***
  - Proximal Policy Optimization

METHOD 1:

Gradient-free neuroevolution with multiple agents

Techniques:

- ***Multiple agents***
  (each agent is a deep neural network)
- ***Model-free approach***
  (no change)
- ***Fitness function***
  - Cumulative reward of an episode
- ***Genetic operators***
  - selection (elitism & tournament)
  - cross-over (n-point)
  - mutation (probability-based)

General process:

A population of agent networks is initialized with random weights. The population of agents are evaluated in an episode of interaction with the environment (a finished schedule). The fitness for each agent is computed as the cumulative sum of the reward they receive over the timesteps in that episode. A selection operator selects a portion of the population for survival with a probability proportional to their relative fitness scores. The agents in the population are then probabilistically perturbed through mutation and crossover operations to create the next generation of agents. k agents with the highest fitness score are preserved as elites and bypass the mutation step. For this method, we only use the evolutionary techniques to evolve the network weights over time making it a gradient-free approach.

METHOD 2:

Gradient-free neuroevolution + Gradient-based optimization with multiple agents

Techniques:

- ***Actor-critic structure***
  (modify to support multiple agents)
- ***Model-free approach***
  (no change)
- ***Off-policy learning***
  (learns the reward function from taking actions using a separate policy (promotes exploration))
- ***Loss function***
  (modify to work with discrete action space)
  - Deep Q-Learning (DQN) or,
  - Deep Deterministic Policy Gradient (DDPG) with Maximum Entropy or,
  - Twin Delay DDPG (TD3) with Gumbel Softmax
- ***Fitness function***
  - Cumulative reward of an episode
- ***Genetic operators***
  - selection (elitism & tournament)
  - cross-over (n-point)
  - mutation (probability-based)

General process:
In order to leverage the benefits of gradient-free and gradient-based optimization a hybrid algorithm will be used. The gradient-free evolutionary process is identical to method 1 and serves as the main objective for our population of actors. The gradient-based optimizer trains policies to maximize agent-specific rewards. These gradient-based policies are periodically added to the evolutionary population and participate in evolution. Additionally, each actor stores its experiences [current state, action, next state, reward] in a replay buffer. This is done for every interaction, at every timestep, for every episode, and for each of its actors. The critic

samples a random minibatch from this replay buffer and uses it to update its parameter using gradient descent. The critic is then used to train a target actor (separate from the population) using the sampled policy gradient. For this method, we employ both gradient-free and gradient-based methods to find the best policy.

METHOD 3:

Neural architecture search with multiple agents

Techniques:

- ***Multiple agents***
  (each agent is a deep neural network)
- ***Model-free approach***
  (no change)
- ***On-policy learning***
  (learns the reward function from taking actions using the current policy)
- ***Loss function***
  - Proximal Policy Optimization
- ***Fitness function***
  - Average reward during training
- ***Genetic operators***
  - selection
  - cross-over
  - mutation

General process:
Instead of evolving the network weights over time like in method 1, we use evolutionary techniques to find a good architecture for our agent. The evaluation process first involves training and optimizing each agent similar to baseline method then computing the fitness for said agent (average reward received during training). The evolutionary process is similar to method 1 but the genetic operators are modified to evolve network architectures instead of weights. These steps are then repeated N times or till a stopping condition is met. While simple on paper, this method might be the most computationally expensive and time-consuming so it will only be explored if the previous methods are found to be insufficient.